

Greedy Awakens: Efficient and Provable Multi-Query Optimization

Tarun Kathuria^{*}
Microsoft Research
Bangalore, India
tarunkathuria@cse.iitb.ac.in

S. Sudarshan
Indian Institute of Technology Bombay
Bombay, India
sudarsha@cse.iitb.ac.in

ABSTRACT

Complex queries for massive data analysis jobs have become increasingly commonplace. Many such queries contain common sub-expressions, either within a single query or among multiple queries submitted as a batch. Conventional query optimizers do not exploit these common sub-expressions and produce sub-optimal plans. The problem of multi-query optimization (MQO) is to generate an optimal *combined* evaluation plan by computing common sub-expressions once and reusing them. Exhaustive algorithms for MQO explore an $\mathcal{O}(n^n)$ search space. Thus, this problem has primarily been tackled using various heuristic algorithms, without providing any theoretical guarantees on the quality of the solution obtained.

In this paper, instead of the conventional cost minimization problem, we treat the problem as maximizing a linear transformation of the cost function. We propose a greedy algorithm for this transformed formulation of the problem, which under weak, intuitive assumptions, gives an approximation factor with respect to the optimal solution of this formulation. We go on to show that this factor is optimal, unless $P = NP$. Another noteworthy point about our algorithm is that it can be easily incorporated into existing transformation-based optimizers. We finally propose optimizations which can be used to improve the efficiency of our algorithm.

Categories and Subject Descriptors

H.2.4 [Database Management]: Systems—*Query processing*

Keywords

Multi-query optimization, greedy algorithms, approximation algorithms

1. INTRODUCTION

Modern data analytics platforms frequently have to run scripts which contain a large number of complex queries. Often, these queries contain common sub-expressions due to the nature of the analysis performed. These sub-expressions may occur within a single complex query which i) contains multiple correlated nested

subqueries or ii) if the database contains many materialized views which are referenced multiple times in the query. A more interesting case where common sub-expressions arise is when a batch of related queries are being executed together.

Conventional query optimizers are not suited for such scenarios since they do not exploit these sub-expressions and instead produce locally optimal plans for each query. These plans can be globally sub-optimal since they do not make use of the shared sub-expressions while generating the plans. The problem of Multi-Query Optimization (MQO) is to generate query plans where these sub-expressions are executed once and their results used by multiple consumers. The selection of the best plan is performed in a completely cost-based manner.

We now present an example to illustrate the MQO problem and how locally optimal plans may lead to globally sub-optimal plans for multiple queries in the presence of common subexpressions.

Example 1.1. (Example 1.1 in [23]) Consider a batch consisting of two queries $(A \bowtie B \bowtie C)$ and $(B \bowtie C \bowtie D)$ whose locally optimal plans (i.e., individual best plans) are $(A \bowtie B) \bowtie C$ and $(B \bowtie C) \bowtie D$ respectively. The individual best plans for the two queries do not have any common sub-expressions. However, consider a locally sub-optimal plan for the first query $A \bowtie (B \bowtie C)$. It is clear that $(B \bowtie C)$ is a common sub-expression and can be computed once and used by both queries.

Consider the following instantiation of the various costs for the two queries shown in Figure 1. Suppose the base relations A , B , C and D each have a scan cost of 10 units. Each of the joins have a cost of 100 units, giving a total evaluation cost of 460 units for the locally optimal plans shown in Figure 1a. On the other hand, in the plan shown in Figure 1b, the common sub-expression $(B \bowtie C)$ is first computed and materialized on the disk at a cost of 10. Then, it is scanned twice - the first time to join with A in order to compute the first query, and the second time to join it with D in order to compute the second - at a cost of 10 per scan. Each of these joins have a cost of 100 units. Thus, the total cost of this consolidated plan is 370 units, which is lesser than the cost of the locally optimal plan in Figure 1a.

It should be noted that blindly sharing a sub-expression

^{*}Majority of the work was done while the author was an undergraduate student at IIT Bombay

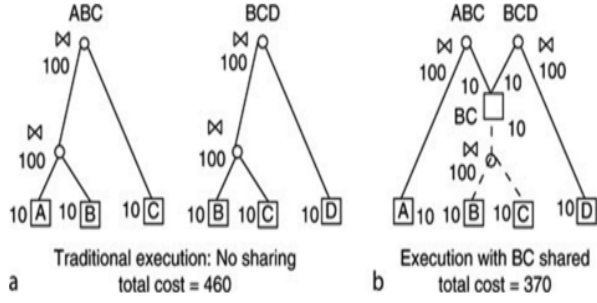


Figure 1: MQO example (from [23]) illustrating benefit of sharing sub-expressions

may not always lead to a globally optimal strategy. For example, there may be cases where the cost of joining the sub-expression $(B \bowtie C)$ with A is very large compared to the cost of the plan $(A \bowtie B) \bowtie C$; in such cases it may make no sense to reuse $(B \bowtie C)$ even if it were available. \square

While algorithms which find the optimal plan for a single query are well known, exhaustive algorithms for MQO take $\mathcal{O}(n^n)$ time which very quickly makes the problem untenable. Thus, work in this area relies on the development of algorithms which are guided by various heuristics [29, 23, 26]. While most of such work has been shown to work well in practice, there has been no work which provides theoretical guarantees on the quality of solution obtained by any such heuristics, to the best of our knowledge. Thus, an open question is

Can we devise a polynomial-time algorithm which provides us with theoretical guarantees on the quality of the solution obtained as compared to the optimal? If so, under some hardness assumption, what is the best possible polynomial-time approximation algorithm?

As a first step towards answering this question, we propose a reformulation of the MQO problem, the motivation for which is stated next.

The canonical multi-query optimization problem is concerned with minimizing the cost of the execution plan for a set of queries by choosing a set of nodes to materialize (say M) and then finding the optimal plan exploiting nodes in M . Another way to look at this problem is to maximize the “materialization-benefit” we get by materializing M w.r.t. a naive execution plan which is locally optimal and does not exploit any common sub-expressions. More formally, this corresponds to maximizing the difference of the cost of the best plan in which the set of materialized nodes is M from the latter. As this is just a linear transformation of the cost function, it is clear that the maximizer of the materialization-benefit function will be the minimizer of the cost function.

Roy et al. [23] assume a property which they call the “monotonicity heuristic” on the cost function. This essentially corresponds to assuming the supermodularity of the cost function defined on the set of nodes to be materialized. In [23], this assumption is used to speed

up their greedy algorithm via a heap-based argument which exploits the supermodularity. This is similar to the LAZYGREEDY algorithm described in [16] for speeding up monotone submodular function maximization subject to cardinality constraints via the well-known greedy algorithm, which is also used by [23]. On the queries used in their experiments, it was observed that the plan obtained with or without assuming supermodularity led to the same plan. This seems to imply that the supermodularity assumption may be a reasonable one and may hold in practice.

1.1 Our contribution

The contributions of this paper are as follows

- Motivated by [23], we proceed with the “monotonicity heuristic” assumption (which implies the submodularity of the materialization benefit function). *Under this assumption*, we propose an approximation algorithm for the underlying problem of unconstrained, normalized submodular maximization (UNSM). Note that we allow the submodular function to take negative values, which has not been considered previously¹.
- We then present a hardness of approximation proof for the UNSM problem, which matches that obtained by our approximation algorithm, under the weak assumption of $P \neq NP$.
- We present optimizations to our algorithm which can be used to improve the running time of the algorithm, without sacrificing any theoretical guarantees.
- We also consider a special case of the problem of submodular maximization under cardinality constraints.
 - A natural extension to our greedy algorithm for this problem is presented. We further propose a pruning strategy to reduce the search space before running our greedy algorithm, by exploiting this cardinality constraint.
 - While, at this point, we do not formally prove any theoretical guarantees on the approximation factor for this constrained problem, we show that the answer obtained by our greedy algorithm is the same when run with or without this pruning.

It is important to note that our approximation guarantees are for the benefit-maximization problem, under the submodularity assumption, and do not imply a multiplicative factor approximation to the cost minimization problem.

Our techniques for the problem of multi-query optimization are presented in the context of query optimizers based on the Volcano/Cascades framework [11, 10].

¹Inapproximability results when the submodular function may be unnormalized are well known.

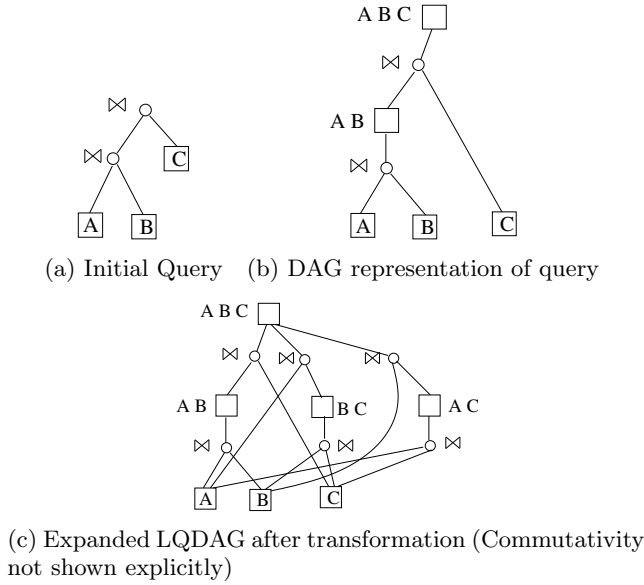


Figure 2: Initial Query and LQDAG Representation

This framework for optimizing queries uses transformation rules which makes it inherently extensible, and has been implemented in several widely-used commercial database systems such as Microsoft SQL Server/SCOPE [4]. It should be noted, however, that our algorithm is agnostic to the query optimization framework and can be easily extended to other frameworks as well.

Organization. In Section 2, we present a detailed overview of multi-query optimization in the context of the Volcano framework which was presented in [23] along with how submodular maximization arises in this context. Section 3 presents our greedy algorithm for unconstrained, normalized submodular maximization with the proof of its approximation factor guarantee. In Section 4, we prove the hardness of approximation of the unconstrained, normalized submodular maximization which rules out better approximation factors than the one attained by our algorithm, under the assumption of $P \neq NP$. Section 5 presents ways to speed up our algorithm. Related work in the areas of MQO and submodular maximization is presented in Section 6. We conclude and discuss directions for future work in Section 7.

2. PRELIMINARIES

This section presents some relevant background in (Multi)-Query Optimization in the Volcano framework followed by some preliminaries of submodular maximization and finally ends with how submodular maximization arises in MQO. Readers well-versed in MQO techniques in Volcano may skip to the third subsection directly.

2.1 Query Optimization in Volcano

The Volcano/Cascades query optimization framework [11, 10] is based on a system of equivalence rules, which specify that the result of a particular transformation of

a query tree is the same as the result of the original query tree. The key aspect of this framework is the efficient implementation of this transformation rules-based approach.

The Volcano framework uses the AND-OR DAG representation [11, 22] for compactly representing the given query and its alternate query plans. An AND-OR DAG is a directed acyclic graph whose nodes can be divided into AND-nodes and OR-nodes; the AND-nodes have only OR-nodes as children and the OR-nodes have only AND-nodes as children. An AND-node corresponds to an algebraic operator, such as the join operator (\bowtie) or a select operator (σ). It represents the expression defined by the operator and its inputs. An OR-node represents a set of logical expressions that generate the same result set; the set of such expressions is defined by the children AND nodes of the OR node, and their inputs. Hereafter, we refer to the OR-nodes and AND-nodes as equivalence nodes and operator nodes respectively.

The given query tree is initially represented in the AND-OR DAG formulation. For example, the query tree of Figure 2a is initially represented in the AND-OR DAG formulation, as shown in Figure 2b. Equivalence nodes are shown as boxes, while operator nodes are shown in circles.

The initial AND-OR DAG is then expanded by applying all possible logical transformations on every node of the initial DAG created from the given query. Suppose the only possible transformations are join associativity and commutativity. Then the plans $A \bowtie (B \bowtie C)$ and $(A \bowtie C) \bowtie B$, as well as several plans equivalent to these, modulo commutativity, can be obtained by transformations on the initial AND-OR DAG of Figure 2b. These are represented in the DAG shown in Figure 2c. The AND-OR DAG representation after applying all the logical transformations is called the (expanded) Logical Query DAG (or LQDAG).

Each operator node can have different physical implementations; for example, a join operator can be implemented as a hash join, a nested loop join or as a merge join. Once the LQDAG has been generated, physical implementation rules are applied on the logical operators to generate the physical AND-OR DAG, which is called the Physical Query DAG or PQDAG for short.

Properties of the results of an expression, such as sort order, that do not form part of the logical data model are called physical properties [11]. The importance of exploiting physical properties such as sort order and partitioning of result sets is well known in traditional optimization. The AND-OR DAG representation considered for MQO actually works on the PQDAG but we present our algorithms to work at the LQDAG level for brevity.

2.2 Multi-Query Optimization in Volcano

This subsection primarily focuses on the techniques presented in [23] for MQO in the Volcano framework. In order to extend the Volcano AND-OR DAG generation for MQO on a batch of queries, the queries are represented together in a single DAG, sharing subex-

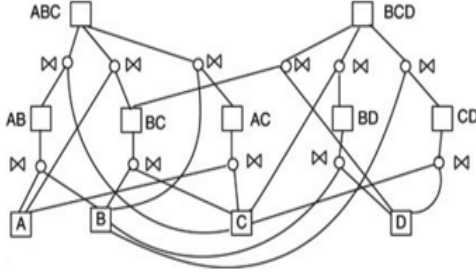


Figure 3: Combined LQDAG for queries in Example 1.1

pressions. The DAG is converted to a rooted DAG by adding a dummy operation node, which does nothing, but has the root equivalence nodes of all the queries as its inputs.

The two main challenges for a multi-query optimizer are :

1. Recognizing possibilities of shared computation by identifying common sub-expressions.
2. Finding a globally optimal evaluation plan exploiting the common sub-expressions identified.

Roy et al. [23] present an efficient algorithm that identifies the set of all common sub-expressions in a single bottom-up traversal of the LQDAG by making use of the “memo” structure used in the Volcano/Cascades framework; for details see [23]. This is similar to the “expression fingerprinting” used to identify the common sub-expressions in [26]. The combined LQDAG for the queries of Example 1.1 is shown in Figure 3.

After the common sub-expressions are identified, the next task is to find the best consolidated plan for the queries exploiting the sub-expressions. In this paper, we are primarily concerned with the optimization philosophy adopted by the Greedy algorithm in [23] which is presented next. For a set of equivalence nodes S , let $bestCost(Q, S)$ (for brevity, $bc(S)^2$) denote the cost of the optimal plan for Q given that nodes in S are to be materialized (this cost includes the cost of computing and materializing nodes in S). Here Q is the combined query DAG with the dummy root operator node with inputs being the DAGs of Q_1, Q_2, \dots, Q_k , as described above.

In [23], they propose an intuitive greedy algorithm which iteratively picks which node to materialize. At each iteration, the node x that gives the maximum reduction in the cost, if materialized, is chosen to be added to the current set of materialized nodes X . While this greedy algorithm is shown to work well in practice, they [23] do not theoretically argue about the quality of solution obtained via this greedy algorithm. The algorithm is presented below for completeness.

As noted in [23], the nodes materialized in the globally optimal plan are just a subset of the ones that are shared in some plan for the query. It is, thus, sufficient to

²Whenever used, the set of queries Q is clear from context.

Algorithm 1 Greedy Algorithm of [23]

```

 $X = \emptyset$ 
 $Y = \text{Set of shareable equivalence nodes in the DAG}$ 
while  $Y \neq \emptyset$  do
    Pick  $x \in Y$  which minimizes  $bc(X \cup \{x\})$ 
    if  $bc(X) > bc(X \cup \{x\})$  then
         $X = X \cup \{x\}, Y = Y \setminus \{x\}$ 
    else
         $Y = \emptyset$ 
    end if
end while
return  $X$ 

```

search only over the set of *shareable* equivalence nodes, instead of searching over the entire set of equivalence nodes in the DAG.

Roy et al. [23] make an additional assumption which they call the “monotonicity heuristic”. Define $benefit(x, X)$ as $bc(X) - bc(X \cup \{x\})$. The assumption is that

$$\forall Y \subseteq X, \forall x \notin X, benefit(x, X) \leq benefit(x, Y).$$

They [23] make this assumption in order to improve the running time of their greedy algorithm via a heap-based argument which corresponds to the LAZYGREEDY algorithm of [16]. Their experiments, however, show that the plans obtained with and without the assumption had exactly the same cost. While the assumption may not always hold, their experiments seem to indicate that the assumption may be a reasonable one, in practice. Thus, in this paper, we work under this assumption to devise an algorithm for maximizing the materialization benefit.

2.3 Submodular Maximization

Let U be a universe of $n = |U|$ elements, let $f : 2^U \rightarrow \mathbb{R}$ be a function. For simplicity, we use the notation $f'(u, S)$ to denote the incremental value in f of adding u to S , i.e., $f'(u, S) = f(S \cup \{u\}) - f(S)$.

Definition 2.1. (SUBMODULAR FUNCTIONS)

A function $f : 2^U \rightarrow \mathbb{R}$ is called *submodular* if

$$\forall A \subseteq B \subseteq U, \forall u \in U \setminus B, \text{ we have } f'(u, A) \geq f'(u, B).$$

Definition 2.2. (SUPERMODULAR FUNCTIONS)

A function $f : 2^U \rightarrow \mathbb{R}$ is called *supermodular* if

$$\forall A \subseteq B \subseteq U, \forall u \in U \setminus B, \text{ we have } f'(u, A) \leq f'(u, B).$$

Definition 2.3. (ADDITIVE FUNCTIONS)

A function $c : 2^U \rightarrow \mathbb{R}$ is called *additive* if it can be represented as $c(S) = \sum_{e \in S} c_e$. Here, $c_e = c(\{e\})$ and we will use both interchangeably.

Definition 2.4. (MONOTONE FUNCTIONS)

A function $f : 2^U \rightarrow \mathbb{R}$ is said to be *monotone* if

$$\forall A \subseteq B \subseteq U, \text{ we have } f(A) \leq f(B).$$

Definition 2.5. (NORMALIZED FUNCTIONS)

A function $f : 2^U \rightarrow \mathbb{R}$ is called *normalized* if $f(\emptyset) = 0$.

Given a normalized submodular function $f : 2^U \rightarrow \mathbb{R}$, the unconstrained, normalized submodular maximization (UNSM) problem is to find a set $S \subseteq U$ which maximizes the value of f . Mathematically, this corresponds to

$$\max_{S \subseteq U} f(S).$$

It is well-known that any non-monotone submodular function f , with the constraint that $f(\emptyset) = 0$, can be written as the difference of a non-negative monotone submodular function f_M and an additive “cost” function c . For completeness, one such decomposition is presented below.

Proposition 2.6. *Any non-monotone (which may take negative values) submodular function f such that $f(\emptyset) = 0$, can be decomposed as*

$$f(S) = f_M(S) - c(S), \forall S \subseteq U$$

where f_M is a monotone submodular function and c is an additive cost function. In particular, one possible decomposition is

$$\begin{aligned} f_M(S) &= f(S) + \sum_{e \in S} (f(U \setminus \{e\}) - f(U)) \\ c(S) &= \sum_{e \in S} (f(U \setminus \{e\}) - f(U)) \end{aligned}$$

Proof. It is easy to see that c is an additive function and that

$$\forall S \subseteq X, \text{ we have } f(S) = f_M(S) - c(S)$$

Since c is additive and f is submodular, it can be easily verified that f_M is submodular as well. Now we just have to show that f_M is monotone.

Consider an arbitrary $S \subset U$ and an arbitrary $e \in U \setminus S$. Let us consider the expression

$$\begin{aligned} & f_M(S \cup \{e\}) - f_M(S) \\ &= f(S \cup \{e\}) - f(S) + (f(U \setminus \{e\}) - f(U)) \\ &= (f(S \cup \{e\}) - f(S)) - (f(U) - f(U \setminus \{e\})) \\ &\geq 0 \end{aligned}$$

The inequality in the last line follows from the fact that $S \subseteq U \setminus \{e\}$ and the submodularity of f . \square

The terms in the summation can be suitably scaled to ensure that c is zero only at \emptyset and positive everywhere else. Note that this implies that the scaled f_M is also non-negative. This follows from the fact that f_M is monotone and $f(\emptyset) = 0$ which implies that $f_M(\emptyset) = 0$.

2.4 Multi-Query Optimization and the UNSM Problem

We now describe the changes to the MQO formulation of [23] and show the role submodularity plays in the same. As defined above, $bestCost(Q, S)$ includes the cost of computing the set of PQDAG nodes to be materialized S . Consider a scenario where S was already materialized and we just have to find the optimal plan

which *may or may not* use the materialized nodes in S . However, no further nodes may be chosen to be materialized. The cost of the optimal plan can be thought of as the *best use benefit* and the function is thus called $bestUseBenefit(Q, S)$. This function is monotonically decreasing since as more nodes are materialized, we will exploit the additional nodes only if they lead to a reduction in cost. Of course, the cost of materializing S needs to be taken into account and we call that function $c(S)$. Clearly, $bestCost(Q, S) = bestUseCost(Q, S) + c(S)$. For brevity, we refer to $bestUseCost(Q, S)$ as $buc(S)$.

The problem of MQO can be thought of as maximizing the “materialization-benefit” ($mb(S)$ for brevity) we can get in the plan cost by exploiting common sub-expressions over a naive execution plan which is just locally optimal and does not exploit sub-expressions. Clearly the cost of the latter is $bc(\emptyset)$ which is the same as $buc(\emptyset)$. Mathematically, $mb(S)$ ³ is defined as

$$\begin{aligned} mb(S) &= buc(\emptyset) - bc(S) \\ &= buc(\emptyset) - (buc(S) + c(S)) \\ &= (buc(\emptyset) - buc(S)) - c(S) \end{aligned}$$

The function in parenthesis in the last line is a monotonically increasing function since $buc(S)$ is a monotonically decreasing function. Also, if the set of materialized nodes S are “far apart” in the PQDAG, the cost of computing and materializing a node $e \in S$ can be thought of as being independent of the other nodes in S . This motivates us to assume that the $c(\cdot)$ function is additive. Of course, this assumption need not be true. For example, if two of the equivalence nodes in S are just below each other, we can significantly benefit by computing the “lower” node and then just reading it from disk to compute the “upper” node. As proved in Proposition 2.6, under the assumption of submodularity, $mb(\cdot)$ can always be decomposed into a difference of monotone, submodular function and an additive function⁴.

Observe that

$$\forall X, \forall x \notin X, benefit(x, X) = -bc'(x, X)$$

Thus, the “monotonicity heuristic” assumption is essentially that the $bestCost$ function is supermodular. This implies that $mb(S)$ is submodular. Note that $mb(\cdot)$ is normalized. Thus, the problem is essentially the UNSM problem with $mb(\cdot)$ as the submodular function.

3. THE MARGINAL GREEDY ALGORITHM FOR UNSM

In this section, we propose a greedy algorithm for the UNSM problem for which we prove an approximation guarantee in this section. A proof of a matching hardness of approximation, under the assumption of $P \neq NP$ is presented in the next section.

³The value of $mb(S)$ can be computed just by invoking $bc(S)$ if we compute $bc(\emptyset)$ at the beginning and store it.

⁴The decomposition in Proposition 2.6 does not actually correspond to the cost of materializing nodes but parallels are drawn for intuition

Given a decomposition of a non-monotone, normalized submodular function f , let the monotone submodular and additive functions be denoted by $f_M(\cdot)$ and $c(\cdot)$. Thus, the problem we want to solve is as follows

$$\max_{S \subseteq U} f(S) = \max_{S \subseteq U} f_M(S) - c(S)$$

The Marginal Greedy algorithm (Algorithm 2) has been proposed before by [28], albeit for non-negative, monotone submodular maximization under knapsack constraints. At each iteration, the algorithm greedily selects the node with the highest use-benefit cost ratio. Sviridenko [28] has to additionally check only those elements which satisfy a budget based on the knapsack constraint which is not required in our case. The algorithm is as follows

Algorithm 2 Marginal Greedy Algorithm

```

 $X = \emptyset$ 
 $Y = \text{Set of shareable equivalence nodes in the DAG}$ 
while  $Y \neq \emptyset$  do
    Pick  $x \in Y$  which maximizes  $r(x, X) = \frac{f'_M(x, X)}{c_x}$ 
    if  $r(x, X) > 1$  then
         $X = X \cup \{x\}, Y = Y \setminus \{x\}$ 
    else
         $Y = \emptyset$ 
    end if
end while
return  $X$ 

```

In the particular case when the given decomposition is the one given in Proposition 2.6, we can compute the term in the summation for each element once and store it. This can be done in just $n + 1$ $bc(S)$ invocations (for the sets U and for $U \setminus \{e_i\} \forall e_i \in V$).

3.1 Approximation Guarantee of Marginal Greedy

Let Θ be an optimal solution. Denote X_i to be the set of nodes selected by the marginal greedy algorithm just after the i^{th} iteration.

Define $\Delta_{f_M}(E, S) = f_M(S \cup E) - f_M(S)$.

We state the main theorem of this section which mentions the approximation guarantee the Marginal Greedy algorithm provides. The approximation factor is not a constant and instead depends on the value of the f and c functions at optimal.

Theorem 3.1. *The answer obtained by the Marginal Greedy algorithm (X) satisfies the following inequality*

$$f(X) \geq \left[1 - \frac{c(\Theta)}{f(\Theta)} \ln \left(1 + \frac{f(\Theta)}{c(\Theta)} \right) \right] f(\Theta).$$

We prove the theorem after presenting a lemma and its corollary which are central to the proof of the theorem. Loosely speaking, the lemma states that upto a certain point in the execution of the Marginal Greedy

algorithm, there exists an element that can be picked and has a marginal benefit-cost ratio which is at least the marginal benefit-cost ratio we would get if we picked all remaining elements in the optimal solution.

Lemma 3.2. *At any iteration $i+1 < n$ in the execution of the marginal greedy algorithm, if $f_M(X_i) < f(\Theta)$, then there exists some element $e \in \Theta \setminus X_i$ that satisfies*

$$\frac{\Delta_{f_M}(\{e\}, X_i)}{c(\{e\})} \geq \frac{\Delta_{f_M}(\Theta, X_i)}{c(\Theta)}.$$

Proof. Firstly, note that if $f_M(X_i) < f(\Theta) = f_M(\Theta) - c(\Theta) \leq f_M(\Theta)$, then $\Theta \setminus X_i \neq \emptyset$. This is because f_M is a monotonically increasing function. Also, note that if S is fixed, $\Delta_{f_M}(E, S)$ is a submodular function in E , due to submodularity of f_M .

We consider two cases. Since the f_M function is monotonically increasing, the numerators on both sides of the inequality are non-negative.

Case 1. $\Delta_{f_M}(\Theta, X_i) = 0$

In this case, the RHS of the inequality is 0. Since the f_M function is monotonically increasing, $\forall e' \in \Theta \setminus X_i$, we have $\frac{\Delta_{f_M}(e', X_i)}{c(\{e'\})} \geq \frac{\Delta_{f_M}(\Theta, X_i)}{c(\Theta)}$. Since $\Theta \setminus X_i \neq \emptyset$, any element $e' \in \Theta \setminus X_i$ satisfies the required inequality.

Case 2. $\Delta_{f_M}(\Theta, X_i) > 0$

We first show that there exists some element $e \in \Theta$ for which the inequality holds. Assume the contradiction, i.e.,

$$\forall e \in \Theta, \frac{\Delta_{f_M}(\{e\}, X_i)}{c(e)} < \frac{\Delta_{f_M}(\Theta, X_i)}{c(\Theta)}.$$

$$\therefore c(e)(\Delta_{f_M}(\Theta, X_i)) > c(\Theta)(\Delta_{f_M}(\{e\}, X_i)).$$

Summing up over all $e \in \Theta$, we get

$$\begin{aligned}
 & \sum_{e \in \Theta} c(e)(\Delta_{f_M}(\Theta, X_i)) > \sum_{e \in \Theta} c(\Theta)(\Delta_{f_M}(\{e\}, X_i)) \\
 \implies & (\Delta_{f_M}(\Theta, X_i)) \sum_{e \in \Theta} c(e) > c(\Theta) \sum_{e \in \Theta} (\Delta_{f_M}(\{e\}, X_i)) \\
 \implies & (\Delta_{f_M}(\Theta, X_i)) c(\Theta) > c(\Theta) \sum_{e \in \Theta} (\Delta_{f_M}(\{e\}, X_i)) \\
 \implies & \Delta_{f_M}(\Theta, X_i) > \sum_{e \in \Theta} (\Delta_{f_M}(\{e\}, X_i)).
 \end{aligned}$$

Since X_i is fixed, from our earlier observation, $\Delta_{f_M}(E, X_i)$ is a submodular function in E . Thus, we have

$$\Delta_{f_M}(\Theta, X_i) \leq \sum_{e \in \Theta} (\Delta_{f_M}(\{e\}, X_i)).$$

This leads to a contradiction. Thus, there exists some element $e' \in \Theta$ for which the required inequality holds.

Now, observe that the RHS of the required inequality in this case is strictly positive and $\forall e \in X_i$, the LHS of the inequality is 0. Hence, $e' \notin X_i$ and we are done. \square

Corollary 3.3. *When the conditions of Lemma 3.2 hold,*

$$\frac{\Delta_{f_M}(\{e\}, X_i) - c(\{e\})}{\Delta_{f_M}(\{e\}, X_i)} \geq \frac{\Delta_{f_M}(\Theta, X_i) - c(\Theta)}{\Delta_{f_M}(\Theta, X_i)}.$$

Proof. From Lemma 3.2, we have

$$\frac{\Delta_{f_M}(\{e\}, X_i)}{c(\{e\})} \geq \frac{\Delta_{f_M}(\Theta, X_i)}{c(\Theta)}.$$

Since f_M is monotonically increasing, it implies

$$\frac{\Delta_{f_M}(\{e\}, X_i) - c_e}{\Delta_{f_M}(\{e\}, X_i)} \geq \frac{\Delta_{f_M}(\Theta, X_i) - c(\Theta)}{\Delta_{f_M}(\Theta, X_i)}.$$

□

Proof. (of Theorem 3.1) Say the marginal greedy algorithm runs for $l \leq n$ iterations. Define $\alpha(X_i)$ to be the rate of increase of f with respect to f_M just after the i^{th} iteration (at thus the current chosen set of elements is X_i). Further, let $e \in U \setminus X_i$ be the next element that will be chosen by the Marginal Greedy algorithm. Note that e is actually a function of X_i and, thus, once X_i is fixed, so is e . Mathematically,

$$\alpha(X_i) = \frac{f(X_i \cup \{e\}) - f(X_i)}{\delta(f_M(X_i))}$$

$$\text{where } \delta(f_M(X_i)) = f_M(X_i \cup \{e\}) - f_M(X_i).$$

Let $j \leq l$ be the maximal index such that $f_M(X_j) < f(\Theta)$. The rate of increase at each iteration i of the algorithm is at least as large as choosing the element from $\Theta \setminus X_i$ with the rate presented in LHS of Corollary 3.3.

The corollary also implies that while $f_M(X_i) < f(\Theta)$, the greedy algorithm has an element that it can pick. This implies that $j < l$. Thus, we have

$$f(X_l) = \sum_{i=0}^{i=l-1} \alpha(X_i) \delta(f_M(X_i)).$$

Using Corollary 3.3,

$$\begin{aligned} f(X_l) &\geq \sum_{i=0}^{i=l-1} \left(\frac{f_M(\Theta) - f_M(X_i) - c(\Theta)}{f_M(\Theta) - f_M(X_i)} \right) \delta(f_M(X_i)) \\ &\geq \sum_{i=0}^{i=l-1} \left(1 - \frac{c(\Theta)}{f_M(\Theta) - f_M(X_i)} \right) \delta(f_M(X_i)). \end{aligned}$$

Since the term in the parenthesis in the last line is a decreasing function of $f_M(X_i)$, we get

$$\begin{aligned} f(X_l) &\geq \int_0^{f_M(X_l)} \left(1 - \frac{c(\Theta)}{f_M(\Theta) - u} \right) du \\ &\geq \int_0^{f(\Theta)} \left(1 - \frac{c(\Theta)}{f_M(\Theta) - u} \right) du \\ &= \left[u + c(\Theta) \ln(f_M(\Theta) - u) \right]_0^{f(\Theta)} \\ &= f(\Theta) + c(\Theta) \ln \left(\frac{f_M(\Theta) - f(\Theta)}{f_M(\Theta)} \right) \\ &= f(\Theta) + c(\Theta) \ln \left(\frac{c(\Theta)}{f(\Theta) + c(\Theta)} \right) \\ &= f(\Theta) - c(\Theta) \ln \left(\frac{c(\Theta) + f(\Theta)}{c(\Theta)} \right) \\ &= f(\Theta) - c(\Theta) \ln \left(1 + \frac{f(\Theta)}{c(\Theta)} \right) \\ &= \left[1 - \frac{c(\Theta)}{f(\Theta)} \ln \left(1 + \frac{f(\Theta)}{c(\Theta)} \right) \right] f(\Theta). \end{aligned}$$

This concludes our proof and gives us our required approximation factor of

$$\left[1 - \frac{c(\Theta)}{f(\Theta)} \ln \left(1 + \frac{f(\Theta)}{c(\Theta)} \right) \right].$$

□

4. INAPPROXIMABILITY OF UNSM

In this section, we prove a hardness of approximation result for the UNSM problem which matches the approximation factor given by the Marginal Greedy algorithm in the previous section.

Theorem 4.1. *For any $\varepsilon > 0$, it is NP-hard to approximate the unconstrained, normalized submodular maximization problem to a factor of at least*

$$\left(1 - \frac{\ln(1 + \gamma)}{\gamma} + \varepsilon \right).$$

Here, $\gamma = \frac{f(\Theta)}{c(\Theta)}$ and Θ is an optimal solution to the unconstrained, normalized submodular maximization problem.

This approximation factor depends on the value at optimal (which may go to 0), implying that a constant factor approximation to the UNSM problem is unlikely.

Before proving Theorem 4.1, we first present a separation result of the Max Coverage problem which is central to the proof of Theorem 4.1.

4.1 Hardness of Approximation of the Set Cover and Max Coverage Problems

An instance $\mathcal{I} = (X, \mathcal{S})$ of the Set Cover problem is defined as follows: we are given the ground set $X = \{e_1, e_2, \dots, e_n\}$ and $\mathcal{S} = \{S_1, S_2, \dots, S_m\} \subseteq 2^X$. The

goal is to choose the minimum number of sets $\mathcal{O} \subseteq \mathcal{S}$ such that $\bigcup_{S_i \in \mathcal{O}} S_i = X$. Feige [8] showed that for

any $\varepsilon > 0$, there is no $(1 - \varepsilon) \ln n$ -approximation polynomial time algorithm for this problem unless $\text{NP} \subseteq \text{DTIME}(n^{O(\log \log n)})$. The hardness was later proved under the weaker assumption of $\text{P} \neq \text{NP}$ by [18, 6].

A problem closely related to the **Set Cover** problem is the **Max Coverage** problem. An instance of the **Max Coverage** problem consists of an instance $\mathcal{I} = (X, \mathcal{S}, l)$ where X is the ground set, \mathcal{S} is a collection of subsets of X , and $l \leq m$ is an integer specifying the budget. The goal is to select l sets $S_{i_1}, S_{i_2}, \dots, S_{i_l}$ and cover as many elements of the ground set as possible. Feige [8] shows that it is **NP-hard** to approximate this problem to within a factor better than $1 - 1/e$.

Krishnaswamy et al. [15] prove a separation result (which is an extension of the **Max Coverage** hardness stated above) which is of interest to us.

Theorem 4.2. (Theorem 2.2 in [15]) Suppose there exists a polynomial algorithm, which for some constants $B \geq 1$ and $\varepsilon > 0$ such that $\varepsilon < e^{-B}$ has the following property : Given any instance (X, \mathcal{S}, l) of **Max Coverage** with optimal value equal to $|X|$ (i.e., there exist l sets that cover the ground set X completely), the algorithm picks a collection of βl sets for some $\beta \in [0, B]$ which can cover $(1 - e^{-\beta} + \varepsilon)n$ elements. Then $\text{P} = \text{NP}$. Note that we allow the algorithm to pick different values of β for different instances of the problem.

Theorem 2.2 in [15] is actually stated under the stronger assumption of $\text{NP} \not\subseteq \text{DTIME}(n^{O(\log \log n)})$. Their reduction relies on the hardness of set cover which, at the time of that paper, was known only under this stronger assumption. Leveraging the set cover hardness result by [18, 6] under the weaker assumption of $\text{P} \neq \text{NP}$, we arrive at Theorem 4.2 without any changes to the proof provided in [15].

Note that the coverage function $f(\mathcal{A}) = \left| \bigcup_{S \in \mathcal{A}} S \right|$ is a monotone, submodular function. The proof of Theorem 4.1 proceeds by considering a special case of the UNSM problem where for a **Max Coverage** instance, $f_M(\mathcal{A})$ is taken to be proportional to the coverage function and the additive cost function $c(\mathcal{A})$ is proportional to the cardinality of the chosen set of subsets \mathcal{A} . We call this the **Profitted Max Coverage** problem.

Problem 4.3. (The **Profitted Max Coverage** problem) An instance of this problem consists of an instance $\mathcal{I} = (X, \mathcal{S}, l)$ like the **Max Coverage** problem. Consider γ to be a constant for this problem whose value will be revealed later.

Let $f_M(\mathcal{A}) = \frac{(\gamma+1)}{\gamma} \frac{\left| \bigcup_{S \in \mathcal{A}} S \right|}{n}$ and $c(\mathcal{A}) = \frac{1}{\gamma} \frac{|\mathcal{A}|}{l}$. The

goal is to maximize

$$\begin{aligned} f(\mathcal{A}) &= f_M(\mathcal{A}) - c(\mathcal{A}) \\ &= \frac{(\gamma+1)}{\gamma} \frac{\left| \bigcup_{S \in \mathcal{A}} S \right|}{n} - \frac{1}{\gamma} \frac{|\mathcal{A}|}{l} \end{aligned}$$

Proof. (of Theorem 4.1) We want to show that if there exists a polynomial time algorithm which approximates the **Profitted Max Coverage** problem to a ratio better than

$$1 - \frac{\ln(\gamma+1)}{\gamma} + \varepsilon \frac{(\gamma+1)}{\gamma},$$

then $\text{P} = \text{NP}$.

We consider an instance $\mathcal{I} = (X, \mathcal{S}, l)$ of the **Max Coverage** problem such that the optimal value is n (i.e., there exist l sets to cover the entire ground set X). Now, let functions f, f_M and c be defined as in Problem 4.3.

[*Completeness*] Let us take a collection of l sets $\mathcal{G} = \{S_{i_1}, S_{i_2}, \dots, S_{i_l}\}$ that cover the ground set X (such a collection exists because \mathcal{I} is a **Max Coverage** instance with optimal value n). The optimal value of the corresponding **Profitted Max Coverage** instance occurs when exactly the sets in \mathcal{G} are chosen.

$$\begin{aligned} f(\mathcal{G}) &= \frac{(\gamma+1)}{\gamma} \frac{n}{n} - \frac{1}{\gamma} \frac{l}{l} \\ &= \frac{(\gamma+1)}{\gamma} - \frac{1}{\gamma} \\ &= 1. \end{aligned}$$

Observe that $\frac{f(\mathcal{G})}{c(\mathcal{G})} = \gamma$.

[*Soundness*] It is easy to see that we will never choose more than $(\gamma+1)l$ sets as the function f will take negative values in those cases.

For any set, say \mathcal{F} , of βl (where $\beta \in [0, \gamma+1]$) subsets from \mathcal{S} which cover at most $(1 - e^{-\beta} + \varepsilon)n$ elements, the value of the **Profitted Max Coverage** instance in this case is at most:

$$\begin{aligned} f(\mathcal{F}) &\leq \frac{(\gamma+1)}{\gamma} \frac{(1 - e^{-\beta} + \varepsilon)n}{n} - \frac{1}{\gamma} \frac{\beta l}{l} \\ &= \frac{(\gamma+1)}{\gamma} (1 - e^{-\beta} + \varepsilon) - \frac{1}{\gamma} \beta \\ &= \frac{(\gamma+1)(1 - e^{-\beta} + \varepsilon) - \beta}{\gamma}. \end{aligned}$$

Differentiating the expression in the last line w.r.t β and setting the derivative to 0, we get

$$\begin{aligned} \frac{\gamma+1}{\gamma} (e^{-\beta}) - \frac{1}{\gamma} &= 0 \\ \implies e^{\beta} &= (\gamma+1) \\ \implies \beta &= \ln(\gamma+1) \leq (\gamma+1). \end{aligned}$$

Thus, the value $f(\mathcal{F})$ is always less than the value attained for that value of β and is

$$f(\mathcal{F}) \leq 1 - \frac{\ln(\gamma+1)}{\gamma} + \varepsilon \frac{(\gamma+1)}{\gamma}.$$

Now, if there exists a polynomial time algorithm (say Alg) which solves the **Profitted Max Coverage** problem to a factor better than $1 - \frac{\ln(\gamma+1)}{\gamma} + \varepsilon \frac{(\gamma+1)}{\gamma}$, then on any input instance of the **Max Coverage** problem such that the optimal value is n , Alg will output a set \mathcal{F} such that $f(\mathcal{F}) > 1 - \frac{\ln(\gamma+1)}{\gamma} + \varepsilon \frac{(\gamma+1)}{\gamma}$ (since the optimal value is 1). Thus, \mathcal{F} covers strictly more than $(1 - e^{-\beta} + \varepsilon)n$ elements with $\beta = \frac{|\mathcal{F}|}{n}$ (by contraposition). By Theorem 4.2, we have $P = NP$.

Hence, we are done. \square

5. SPEEDING UP THE MARGINAL GREEDY

In the worst case, the Marginal Greedy algorithm runs in $\mathcal{O}(n^2)$ time, where n is the number of shareable nodes in the PQDAG. This makes the algorithm very expensive since n itself may be exponential in the worst case. Thus, we would like to reduce the time taken by the algorithm without sacrificing on the theoretical guarantees on the quality of the solution proved in Section 3. In this section, we present some optimizations to our algorithm to improve the running time of the algorithm.

5.1 Basic Optimizations

We first note that the three optimizations presented in [23] can be used for our algorithm as well. The first observation in their paper, as already mentioned is about searching over all the shareable nodes. As noted above, this can be directly used by us since our algorithm just presents a different heuristic for choosing which nodes to materialize. Their second optimization presents a way to incrementally update the *best-Cost* function for various sets that exploits the result of earlier cost computations to incrementally compute the new plan. Since the $mb(\cdot)$ function is just a linear transformation of the *bestCost* function and our greedy algorithm (at least when the decomposition presented in the proof of Proposition 2.6 is used) is also concerned with just successive differences in the values of the *best-Cost* function, their optimization can also be used to speed up our algorithm; for details see [23].

Another small optimization (not in [23]) that can be made is based on a simple observation of the greedy algorithm and by exploiting submodularity. In the i^{th} iteration, the marginal greedy algorithm needs to compute the maximum benefit-cost ratio $\frac{f'_M(e, X_{i-1})}{c_e}$. Thus, if while scanning elements to compute the maximum, we encounter an element which has the marginal benefit-cost ratio less than 1, we can remove it from the set Y of elements to be searched over as it will never be picked by the marginal greedy algorithm in the future iterations either. This is because f_M is also submodular and the size of X_i always increases as i increases so the value of the marginal benefit-cost ratio only decreases as the algorithm proceeds and will never become greater than 1. We note that a similar optimization for the simple greedy algorithm used for monotone, submodular maximization under cardinality constraints is also possible.

5.2 The LAZYMARGINALGREEDY algorithm

The third optimization in [23] essentially leverages supermodularity to improve the running time of the complexity. The argument is similar to that used by [16] for the LAZYGREEDY algorithm. We observe that a similar argument as the ones presented in these two papers may be used for the Marginal Greedy algorithm and is presented next.

As noted previously, in each iteration i , the Marginal Greedy algorithm must identify the element e with the maximum marginal benefit-cost ratio $\frac{f'_M(e, X_{i-1})}{c_e}$. For each element e , the denominator is fixed and the marginal benefits are monotonically nonincreasing during the iterations of the algorithm, i.e., $f'_M(e, X_i) \geq f'_M(e, X_j)$ whenever $i \leq j$. Thus, instead of recomputing $\frac{f'_M(e, X_{i-1})}{c_e}$ for each element $e \in V$, which requires $\mathcal{O}(n)$ computations of f , the LAZYMARGINALGREEDY algorithm maintains a list of upper bounds $u(e)$ (initialized to a very large value) on the marginal benefit-cost ration sorted in decreasing order (done via a heap).

In each iteration, the algorithm extracts the maximal element from the ordered list of remaining elements. If, after this update, $u(e) \geq u(e') \forall e' \neq e$, then submodularity guarantees that $\frac{f'_M(e, X_{i-1})}{c_e} \geq \frac{f'_M(e', X_{i-1})}{c_{e'}} \forall e' \neq e$, and therefore the algorithm has identified the element with the largest marginal benefit-cost ratio without having to compute the ratio for a potentially large number of elements e' .

5.3 Ground Set Reduction under cardinality constraints

We may sometimes want to consider a cardinality constraint (say k) on the number of nodes to be materialized. This may arise when there are storage constraints which only allow us to materialize a certain number of sub-expressions. We adapt our greedy algorithm for this constraint by simply stopping after k elements are picked.

While, at this point, we do not show any theoretical approximation guarantees for this problem, there is a way to leverage this cardinality constraint to prune out certain elements from the ground set U . This preprocessing step may be used to reduce the size of the set of PQDAG nodes U on which the algorithm will be run.

We show that the algorithm run on this reduced set (say U') is the same as that obtained when the algorithm runs over the full set. This check is useful only when we have chosen to have a cardinality constraint of $k < n$ as we will show.

Theorem 5.1. *Let $U = \{e_1, e_2, \dots, e_n\}$ be the set of all shareable PQDAG nodes ordered as $\frac{f'_M(e_1, U \setminus \{e_1\})}{c_{e_1}} \geq \frac{f'_M(e_2, U \setminus \{e_2\})}{c_{e_2}} \geq \dots \geq \frac{f'_M(e_n, U \setminus \{e_n\})}{c_{e_n}}$. Furthermore, let $U' = \{e \in U \mid \frac{f_M(e)}{c_e} \geq \frac{f'_M(e_k, U \setminus \{e_k\})}{c_{e_k}}\}$ for $k < n$.*

The output of the Marginal Greedy algorithm (with cardinality constraint of k) when it runs on U is the same

as the output when it runs on U' .

Proof. Without loss of generality, assume that the algorithm, when run on V , terminates after the full k steps. Let the sequence of chosen elements, in order of inclusion, be $\{s_1, s_2, \dots, s_k\}$ and let $X_i = \{s_1, s_2, \dots, s_i\} \forall i \in [k]$, as before. Clearly, $\emptyset = X_0 \subset X_1 \subset X_2 \subset \dots \subset X_k$.

Case 1. $k = n$

This is a simple case in which all elements are chosen and, thus, U' should be equal to U which is shown as follows $\forall e \in U$, we have

$$\begin{aligned} \frac{f_M(\{e\})}{c_e} &= \frac{f'_M(e, \emptyset)}{c_e} \\ &\geq \frac{f'_M(e, U \setminus \{e\})}{c_e} \quad (\text{by submodularity}) \\ &\geq \frac{f'_M(e_k, U \setminus \{e_k\})}{c_{e_k}}. \end{aligned}$$

Hence, all elements of U are going to be in U' since they all satisfy the condition to be in U' . In this case, the check is clearly wasteful since the ground set has no reduction and a lot of functional calls are made. In the MQO context, this corresponds to invoking a lot of $\text{bestCost}(Q, S)$ calls, each of which are moderately expensive. Thus, in this case, the preprocessing step should just check if $k = n$ and if so, directly pass the full ground set to the greedy algorithm.

Case 2. $k < n$ & $X_k = \{e_1, e_2, \dots, e_k\}$.

In this case, the theorem follows trivially since U' will contain all elements in X_k , along with some other elements.

Case 3. $k < n$ & $X_k \neq \{e_1, e_2, \dots, e_k\}$.

We first make a claim which we will prove later.

Claim 5.2. For all $i \in \{1, 2, \dots, k\}$, we have

$$\frac{f'_M(s_i, X_{i-1})}{c_{s_i}} \geq \frac{f'_M(e_i, U \setminus \{e_i\})}{c_{e_i}}.$$

The claim is used to show that elements in $U \setminus U'$ will never be picked by the marginal greedy algorithm. Intuitively, for any element $e \notin U'$, the $\frac{f'_M(e, X_i)}{c_e}$ ratio of picking it is largest in the first iteration (by submodularity) and that itself is less than the element with the smallest ratio of the elements selected by the greedy algorithm. So, it is guaranteed that the greedy algorithm does not pick any element which is not in U' . This is easy to see and is as follows
 $\forall e \in U \setminus U'$, we have

$$\frac{f'_M(e, \emptyset)}{c_e} = \frac{f_M(e)}{c_e} < \frac{f'_M(e_k, U \setminus \{e_k\})}{c_{e_k}}.$$

By Claim 5.2,

$$\begin{aligned} \frac{f'_M(s_k, X_{k-1})}{c_{s_k}} &\geq \frac{f'_M(e_k, U \setminus \{e_k\})}{c_{e_k}} \\ \implies \frac{f'_M(s_k, X_{k-1})}{c_{s_k}} &> \frac{f'_M(e, \emptyset)}{c_e}. \end{aligned}$$

We now present the proof of Claim 5.2.

Proof. (of Claim 5.2) $e_i \in X_{i-1}$

Since $|X_{i-1}| = i - 1$, X_{i-1} cannot include all elements from the set $\{e_1, e_2, \dots, e_i\}$. Thus, there exists some element, say, $e_z \in e_1, e_2, \dots, e_i$ such that $e_z \notin X_{i-1}$.

$$\begin{aligned} \text{Thus, we have } \frac{f'_M(s_i, X_{i-1})}{c_{s_i}} &= \max_{e \in U \setminus X_{i-1}} \frac{f'_M(e, X_{i-1})}{c_e} \\ &\geq \frac{f'_M(e_z, X_{i-1})}{c_{e_z}} \\ &\geq \frac{f'_M(e_z, U \setminus \{e_z\})}{c_{e_z}} \quad (\text{by submodularity}) \\ &\geq \frac{f'_M(e_i, U \setminus \{e_i\})}{c_{e_i}} \end{aligned}$$

□

This concludes our proof. □

It is important to note that this strategy may not always lead to a reduction in the ground set but it may lead to pruning in certain cases.

Note that this pruning procedure can be modified to work for the simple greedy algorithm for monotone, submodular maximization under cardinality constraints. The proof is also along similar lines as those stated above.

6. RELATED WORK

We now present the related work in the areas of multi-query optimization and submodular maximization.

6.1 Multi-Query Optimization

The problem of multi-query optimization has received significant amount of attention in the past [24, 20, 25, 21, 27]. The initial work by [24, 20, 21, 25] proposed solutions that were not fully integrated with the query optimizer and utilized primarily exhaustive algorithms.

Shim et al. [25] consider heuristics to reduce the cost of multi-query optimization. However, even with heuristics, these approaches are extremely expensive for situations where each query may have a large number of alternative evaluation plans. Subramanian and Venkataraman [27] consider sharing only among the best plans of the query; this approach can be implemented as an efficient, post-optimization phase in existing systems, but does not guarantee optimality, as discussed above.

To choose the set of nodes to be materialized, Roy et al. [23] use a greedy algorithm which has already been discussed in detail in Section 2. Dalvi et al. [5] explores the possibility of sharing intermediate results by pipelining, avoiding unnecessary materializations. Diwan et al. [7] consider the generalized problem of MQO in Volcano which also takes scheduling and caching into account. They present an exhaustive algorithm which takes $\mathcal{O}(n^n)$ time, which is clearly infeasible.

Zhou et al. [29] propose a framework to use common subexpressions for MQO and materialized view selec-

tion in a query optimizer based on the Cascades framework [10]. The focus however is on “covering” subexpressions at the LQDAG level and does not take into account competing physical properties like sort orders and partitioning properties from different consumers.

Silva et al. [26] consider physical properties in a cost-based fashion. However, their solution is also based on heuristics which choose to materialize every common sub-expression at the LQDAG level. The best physical property for each sub-expression is chosen and all consumers are forced to use the same physical property, which can be sub-optimal. Furthermore, even with this heuristic, their approach can be very expensive in cases where there are potentially many physical properties for each common sub-expression.

6.2 Submodular Maximization

Submodular maximization has received a significant amount of attention in optimization [3, 19, 2] and has a wide variety of applications in machine learning and computer vision [12, 13, 1, 14] and even speeding up satisfiability solvers [9]. In this problem, we are given a submodular function f and a universe U , with the goal of selecting a subset $S \subseteq U$ such that $f(S)$ is maximized. Typically, S must satisfy additional feasibility constraints such as cardinality, knapsack or matroid constraints.

This problem is NP-hard even for the simplest problems which involve only *cardinality constraints* and monotone functions. Nemhauser et al. [19] show that a simple greedy algorithm gives a $(1 - 1/e)$ approximation for monotone submodular maximization under cardinality constraints. They further show that it is NP-hard to obtain a better approximation guarantee. In [3], they prove the same approximation guarantee for monotone submodular maximization under special matroid constraints, albeit via a different algorithm. Sviridenko [28] presents a modified greedy algorithm for monotone submodular function maximization under knapsack constraints. Their algorithm is the main motivation for our marginal greedy algorithm.

Buchbinder et al. [2] gave a $1/2$ -approximation algorithm for unconstrained non-monotone submodular maximization, for which there is a matching hardness result. However, all these results assume non-negativity of the function f . Mittal and Shulz [17] show that a constant factor approximation for non-negative supermodular minimization is NP-hard. Inapproximability of non-monotone submodular maximization (with possibly negative values) is also well known. To the best of our knowledge, ours is the first work which, under the assumption of $f(\emptyset) = 0$, provides an approximation algorithm with a matching hardness of approximation result for unconstrained non-monotone submodular maximization when the function may take negative values. Since the hardness of approximation factor depends on the optimal (and may go to 0), this rules out constant factor approximations for the problem even in the restricted setting of $f(\emptyset) = 0$.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a reformulation of the well-studied MQO problem. Under the assumption of supermodularity of the *bestCost* function, we propose a greedy algorithm for the maximization problem. We give an approximation factor guarantee for our algorithm. We have then shown that obtaining a better approximation factor than the one attained by our greedy algorithm is NP-hard. Such theoretical guarantees on the quality of any heuristic has not been presented before. Since the underlying problem solved in this paper is the unconstrained, normalized submodular maximization problem (with possibly negative values), we believe our results can be useful beyond just the MQO problem.

One area of future work is the problem of submodular maximization problem (with possibly negative values) under cardinality constraints. We would like to see if our adapted greedy algorithm gives any approximation factor guarantees as well as a hardness of approximation proof for the problem. Another important line of work is to implement our greedy algorithm and compare its performance with other multi-query optimizers which are currently being used.

8. ACKNOWLEDGMENTS

The authors would like to thank Amit Deshpande and Deeparnab Chakrabarty for pointing to the relevant background material. We would like to also thank them, along with Ravishankar Krishnaswamy for comments about the hardness of approximation proof presented above.

9. REFERENCES

- [1] Y. Boykov and M. Jolly. Interactive Graph Cuts for Optimal Boundary & Region Segmentation of Objects in N-D Images. *International Conference on Computer Vision (ICCV)*, 1(July):105–112, 2001.
- [2] N. Buchbinder, M. Feldman, J. Naor, and R. Schwartz. A tight linear time $(1/2)$ -approximation for unconstrained submodular maximization. *Foundations of Computer Science (FOCS)*, pages 649–658, 2012.
- [3] G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, Dec. 2011.
- [4] R. Chaiken, B. Jenkins, P.-å. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. SCOPE : Easy and Efficient Parallel Processing of Massive Data Sets. *VLDB*, 2008.
- [5] N. N. Dalvi, S. K. Sanghai, P. Roy, and S. Sudarshan. Pipelining in multi-query optimization. *Journal of Computer and System Sciences*, 66:728–762, 2003.
- [6] I. Dinur and D. Steurer. Analytical approach to parallel repetition. In *Symposium on Theory of*

- Computing*, STOC '14, pages 624–633, New York, NY, USA, 2014. ACM.
- [7] A. A. Diwan, S. Sudarshan, and D. Thomas. Scheduling and Caching in Multi-Query Optimization. *International Conference on Management of Data (COMAD)*, pages 14–17, 2006.
 - [8] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.
 - [9] D. M. S. Golovin. An online approach to maximizing submodular functions. *Neural Information Processing Systems*, pages 1–8, 2008.
 - [10] G. Graefe. The Cascades framework for query optimization. *IEEE Data Engineering Bulletin*, 18(3):19–29, 1995.
 - [11] G. Graefe and W. McKenna. The Volcano optimizer generator: extensibility and efficient search. *International Conference on Data Engineering (ICDE)*, pages 209–218, 1993.
 - [12] S. Jegelka and J. A. Bilmes. Submodularity beyond submodular energies: Coupling edges in graph cuts. In *CVPR 2011*, pages 1897–1904, 2011.
 - [13] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. *KDD*, page 137, 2003.
 - [14] P. Kohli, M. P. Kumar, and P. H. S. Torr. P3 & beyond: Move making algorithms for solving higher order functions. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(9):1645–1656, 2009.
 - [15] R. Krishnaswamy and M. Sviridenko. Inapproximability of the Multi-level Uncapacitated Facility Location Problem. *Symposium on Discrete Algorithms (SODA)*, pages 718–734, 2012.
 - [16] M. Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In J. Stoer, editor, *Optimization Techniques*, volume 7 of *Lecture Notes in Control and Information Sciences*, pages 234–243. Springer Berlin Heidelberg, 1978.
 - [17] S. Mittal and A. S. Schulz. An FPTAS for optimizing a class of low-rank functions over a polytope. *Math. Program.*, 141(1-2):103–120, 2013.
 - [18] D. Moshkovitz. The Projection Games Conjecture and the NP-Hardness of $\ln n$ -Approximating Set-Cover. *Theory of Computing*, 11(1):221–235, 2015.
 - [19] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions-I. *Mathematical Programming*, 14(1):265–294, 1978.
 - [20] J. Park and A. Segev. Using common subexpressions to optimize multiple queries. In *International Conference on Data Engineering (ICDE)*, pages 311–319, 1988.
 - [21] A. Rosenthal and U. Chakravarthy. Anatomy of a modular multiple query optimizer. *Very Large Data Bases (VLDB)*, pages 230–239, 1988.
 - [22] N. Roussopoulos. View indexing in relational databases. *ACM Trans. Database Syst.*, 7(2):258–290, June 1982.
 - [23] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhohe. Efficient and Extensible Algorithms for Multi Query Optimization. *SIGMOD*, pages 249–260, 2000.
 - [24] T. K. Sellis. Multiple-query optimization. *ACM Trans. Database Syst.*, 13(1):23–52, Mar. 1988.
 - [25] K. Shim, T. Sellis, and D. Nau. Improvements on a heuristic algorithm for multiple-query optimization. *Data & Knowledge Engineering*, 12:197–222, 1994.
 - [26] Y. N. Silva, P. A. Larson, and J. Zhou. Exploiting Common Subexpressions for Cloud Query Processing. *International Conference on Data Engineering (ICDE)*, 1, 2012.
 - [27] S. N. Subramanian and S. Venkataraman. Cost-Based Optimization of Decision Support Queries using. *SIGMOD '98*, pages 319–330, 1998.
 - [28] M. Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Oper. Res. Lett.*, 32(1):41–43, 2004.
 - [29] J. Zhou, P. A. Larson, J. C. Freytag, and W. Lehner. Efficient exploitation of similar subexpressions for query processing. *SIGMOD*, pages 533–544, 2007.